Using OpenACC Compilers to Run FIM and NIM on GPUs

Mark Govett

NOAA Earth System Research Laboratory

Background

- Developing NIM model to be highly scalable a single source code and performance portable
  - Intel SB, Xeon Phi, Kepler GPU, etc
- Plan to run NIM at 3.5KM resolution in 2014
  - Require a minimum of 2000 GPUs or Xeon-Phi
  - Further work to optimize communications and compute
- First of several models projected to run on Fine-Grain computers in 2014 & 2015
  - FIM, HRRR (variant of WRF-ARW)

Goals of this Talk

- Report on our recent experience with the openACC compilers from PGI, Cray, CAPS
  - Determine ability to handle FIM, NIM codes without code change
  - Maintain performance portability to CPU, GPU, MIC
- Compare performance to F2C-ACC

MIC & GPU Performance for NIM

- 10242 horizontal points, 96 vertical levels
- Kepler results used the F2C-ACC compiler
- Xeon Phi are from the 7110 version on TACC
- The source code for these runs is the identical
  - New diag code has been optimized for MIC, but not run on the Kepler

Main Routines	Kepler K20x	Xeon Phi
TOTAL: Main loop	[19.61]	[20.73]
vdmints	6.29	7.02
vdmintv	2.43	3.44
flux	1.11	1.79
trisol	0.76	0.52
force	0.64	0.93
Vdn	0.50	0.93
Diag	[2.70]	[1.11*]

F2C-ACC Compiler

- Developed in 2008 before commercial compilers were available
- Developed for FIM and NIM
  - New capabilities added as needed
  - Used primarily for model dynamics
- Limited Capabilities, Scope, Support
  - Partial support for Fortran 90
  - Shared with a few outside groups
  - No attempt to conform to openACC standard
  - No new development since the last workshop
- Evaluate commercial compilers
  - Share results, code with vendors
  - Henderson, 2010: pre-OpenACC CAPS, PGI compilers
  - Govett, 2013

Using the OpenACC Compilers

- Simple, easy to use
- Feedback by compilers was useful
- Placement of directives was trivial
  - Same as F2C-ACC, similar to openMP
- OpenACC directives required less information than F2C-ACC to prescribe parallelism
- OpenACC compilers have more capabilities than F2C

Directives to Identify Parallel Regions

- F2C-ACC
  - !ACC$REGION (<threads>, <blocks>
    [, <data movement>]
  - !ACC$REGION END

- OpenACC
  - !$acc kernels
  - !$acc end kernels

  - !$acc parallel [num_gangs][num_workers][vector_length]
    [data movement]
  - !$acc end parallel

Directives for Loops

- F2C-ACC
  - !ACC$DO [PARALLEL] [VECTOR]
- OpenACC
  - !$acc loop [gang] [worker] [vector]

Directives for Data Movement

- F2C-ACC
  - !ACC$REGION, !ACC$DATA
- OpenACC
  - !$acc parallel,   !$acc data !$acc update

Additional Directives

- F2C-ACC
  - !ACC$ROUTINE
  - !ACC$THREAD
- OpenACC
  - !$acc routine
  - !$acc async
  - !$acc cache
  - !$acc declare
  - !$acc wait

Standalone Test Cases

- 5 tests from FIM, NIM, WRF

– Cnuity	FIM	Dynamics
– Trcadv	FIM	Dynamics
– WRF-PBL	WRF	Physics
– Momtum	FIM	Dynamics
– Vdmintv	NIM	Dynamics

- Tests run on Titan using F2C-ACC, PGI & Cray
- OpenACC directives used
  - !$acc kernels
  - !$acc parallel [num_gangs] [num_workers] [vector_length]
  - !$acc loop [gang] [vector]
- Share results with vendors
  - Collaborate on tuning

Bitwise Exact Results

- Important to validate the parallelization
  - Speeds parallelization, leaves no doubt with scientists
- Correctness may be needed for long simulations
- Compiler options needed to generate correct results
  - Can be turned off for speed, but essential for correctness

F2C-ACC w/ Intel, CUDA compiler

```
Diff stats for tr3d_ref vs. tr3d:


tracer=         1
   0 diffs were found of a possible      655488
tracer=         2
   0 diffs were found of a possible      655488
tracer=         3
   0 diffs were found of a possible      655488
tracer=         4
   0 diffs were found of a possible      655488
```

Cray, PGI compilers

```
Diff stats for tr3d_ref vs. tr3d:

tracer=         1
   1551 diffs were found of a possible      655488
max diff=          2.4414063E-04  at k,ipn=          64        3285
arr1,arr2=          1996.102        1996.103
number of decimal digits=   6.9
max relative diff=   2.1192625E-07  at k,ipn=          15        1654
arr1,arr2=          288.0019        288.0018
number of decimal digits=   6.7
average number of matching digits for points with diffs=   7.0
tracer=         2
   10807 diffs were found of a possible      655488
```

FIM Dynamics: CNUITY

```
!ACC$REGION(<64>,<10242>,<flxhi:none,local>) BEGIN
!$acc parallel num_gangs(ihe-ips+1) vector_length(64) private(flxhi)
!ACC$DO PARALLEL(1)
!$acc loop gang
    do ipn=ips,ihe
!ACC$DO VECTOR(1)
!$acc loop vector
      do k=1,nvl
        flxhi(k) = vnorm(k,edg,ipn)*dp_edg(k,edg,ipn)
        massfx(k,edg,ipn,nf) = 0.5*((vnorm(k,edg,ipn) +
        abs(vnorm(k,edg,ipn)))*delp(k,ipn) - &
        (vnorm(k,edx,ipx) + abs(vnorm(k,edx,ipx)))*delp(k,ipx))
       enddo
     enddo
!$acc end parallel
!ACC$REGION END
```

- FIM has 64 vertical levels, 10242 horizontal points / GPU
- One horizontal dimension – ipn
- One vertical dimension - k

FIM Dynamics: CNUITY

Compiler	Cnuity1	Cnuity2	Cnuity3	Cnuity4	cnuity5	Total (% slower)
F2C	325, 514	295	421	13, 1024	3552	6114
PGI: parallel	616, 993	463	798	159, 1286	8941	13256 (116%)
Cray: parallel	682, 1739	745	964	43, 1893	4031	10097 (65%)
Cray: fast32	555, 979	432	748	37, 1322	3903	7976 (30%)

- Compiler versions & settings used
  - CRAY V8.1.9: -O3 -h noomp,acc -em -ef -eZ –ra
    - -fast_addr option uses 32 bit addressing for array references
  - PGI V13.7.0: -O3 –acc –Minfo=accel
    - -ta=nvidia:cuda5.0 sometimes yielded 10-20% faster runtimes
- Kernel execution times for Kepler K20x
  - Does not include data movement
- Code modified for PGI due to a bug with handling "private" (GPU local) variables resulted in a significant performance penalty
- Both !$acc parallel and !$acc kernels were tried
  - !$acc parallel <u>almost</u> always faster than !$acc kernels

FIM Dynamics: TRCADV

- 64 vertical levels, 10242 horizontal points
- Computation time only – does not include data movement

Compiler	Trcadv1	Trcadv2	Trcadv3	Total(% slower)
F2C-ACC	1031	463	679	2173
PGI	1615	885	871	3371 (55%)
Cray	2895	757	1233	4885 (124%)
Cray-fast32	1205	607	1076	2888 (32%)

Code Example: WRF-PBL

```
!$acc parallel num_gangs((ite-its+1)/64+1) vector_length(64)
 do k = kts,kte                          !vertical dimension loop
!$acc loop gang vector
     do i = its,ite                      !horizontal dimension loop
       zq(i,k+1) = dz8w2d(i,k)+zq(i,k)
     enddo
   enddo
!
   do k = kts,kte
!$acc loop gang vector
     do i = its,ite
       za(i,k) = 0.5*(zq(i,k)+zq(i,k+1))
       dzq(i,k) = zq(i,k+1)-zq(i,k)
       del(i,k) = p2di(i,k)-p2di(i,k+1)
     enddo
   enddo
```

- Typical loop structure for WRF physics
- Apply block (gang) and thread (vector) level parallelization to a single dimension for NIM
  - Dependence on “k” prevents parallelization

WRF Physics: PBL

Compiler	Kernel1	Kernel2	Kernel3	Kernel4	Kernel5	Kernel6	TOTAL
F2C	178	197	665	703	613	91	2447
Cray	250	281	[740]	[1090]	[699]	87	3147
PGI	351	[301]	[829]	652, BUG	[731]	84	NA
Cray: fast	193	202	[613]	[780]	[566]	75	2494

- WRF Version 3.3 code with minor changes
  - F2C-ACC has limited Fortran support
  - To get bitwise exact results compared to the CPU
    - calculations with **2 modified to multiply factors directly
- Square brackets indicate more than one kernel was used

Why are the openACC Compilers Slower?

- Use of Memory: Local, Shared, Global, Registers?
  - Significant benefit using private (thread local) memory observed by all compilers
  - Minimal benefit using shared memory (F2C), did not test with Cray, PGI
  - 10-30% more registers used in Cray, PGI than F2C-ACC
- Parallelism: Increasing the number of threads / block from 64 – 128
  - 32% performance improvements for F2C-ACC routines observed
  - Degraded performance for Cray, PGI compilers observed
    - No combination of gangs, workers, vectors yielded benefit

Compiler - threads	**Cnuity1**	**Cnuity2**	**Cnuity3**	**Cnuity4**	**cnuity5**	**% benefit**
F2C – 64	, 701	400	506	, 1384		
F2C – 128	325, 514	295	421	13, 1024	3552	32% faster
Cray – 64	555, 980	431	750	13, 1322	3895	
Cray – 128	,1272	457	974	, 1419		18% slower
PGI – 64	616, 993	463	798	159, 1286	8941	
PGI – 128	, 1094	480	977	, 1403		11% slower

Summary

- Goal is to have performance benchmarks of the FIM and NIM in the next few months
  - Both physics and dynamics if possible
- OpenACC compilers are the future for GPU programming
  - Standard is sufficient for our applications
  - Easy to use, parallelization is straightforward
  - No code changes required for parallelization
    - Except for the PGI handling of "private" variables
- We will work with vendors to improve performance
  - Modify our use of the openACC directives
    - are we missing something?
  - Provide stand-alone tests to vendors on our website for profiling and analysis